FastBit: Interactively Searching Massive Data

Kesheng Wu[*], Sean Ahern[†], E. Wes Bethel[‡], Jacqueline Chen[§], Hank Childs[¶],

Estelle Cormier-Michel[*], Cameron Geddes[*], Junmin Gu[*], Hans Hagen[‖],

Bernd Hamann[‡], Wendy Koegler[§], Jerome Lauret[**], Jeremy Meredith[†],

Peter Messmer[††], Ekow Otoo[*], Victor Perevoztchikov[**], Arthur Poskanzer[*],

Prabhat[*], Oliver Rübel[‡], Arie Shoshani[*], Alexander Sim[*], Kurt Stockinger[*],

Gunther Weber[‡], Wei-Ming Zhang[‡‡]

June 2009

Abstract

As scientific instruments and computer simulations produce more and more data, the task of locating the essential information to gain insight becomes increasingly difficult. FastBit is an efficient software tool to address this challenge. In this article, we present a summary of the key underlying technologies, namely bitmap compression, encoding, and binning. Together these techniques enable FastBit to answer structured (SQL) queries orders of magnitude faster than popular database systems. To illustrate how FastBit is used in applications, we present three examples involving a high-energy physics experiment, a combustion simulation, and an accelerator simulation. In each case, FastBit significantly reduces the response time and enables interactive exploration on terabytes of data.

1 Introduction

Current generations of scientific instruments [1, 11] and computer simulations [6, 12] are capable of producing massive amounts of data. However, a relatively small percentage of records often hold the key to the insight sought by scientists. For example, the high-energy physics experiment STAR [14] collects billions of events of colliding particles, however, only a few hundred of them might be the rare events having the unique signature of Quark-Gluon Plasma – finding evidence of Quark-Gluon Plasma is one of the key scientific objectives of STAR. Locating a relatively small number of data records distributed throughout a huge collection is a common challenge in many scientific fields. Database Management Systems (DBMS) are designed to handle such tasks, however, existing DBMS are not well-suited for scientific data explorations.

The existing DBMS systems are designed for transactional applications where data records are frequently modified such as modifying one's bank account during an ATM withdrawal. In contrast, scientific applications typically do not perform such transactions. Most interactions with a scientific

---

[*]Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA.

[†]Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA.

[‡]University of California - Davis, Davis, CA 95616, USA.

[§]Sandia National Laboratories, Livermore, CA 94551, USA.

[¶]Lawrence Livermore National Laboratory, Livermore, CA 94550, USA.

[‖]Technische Universität Kaiserslautern, Erwin-Schrödinger-Straße, Kaiserslautern, Germany.

[**]Brookhaven National Laboratory, Upton, NY 11973, USA.

[††]Tech-X Corporation, Boulder, CO 80303, USA.

[‡‡]Kent State University, Kent, OH 44240, USA.

data set are to select a subset of records for further analyses, for example, selecting a number of collision events with certain properties and computing a histogram of their energy distribution, or selecting regions with a high temperature in a combustion simulation and computing the amount of heat generated from each region. Such analysis tasks are generally supported by a data warehousing system and require a different data access technology [5, 10, 15].

A familiar search technology that routinely handles a large amount of data is the web search engine [3, 43]. The key underlying technology is the inverted index designed for text data. Since the bulk of scientific data are numerical values, this indexing technology is not applicable. However, other technologies employed by search engines such as the parallel iterators can be used effectively for scientific applications.

We have been working on a set of techniques specifically designed for scientific data. They are based on the bitmap index [4, 18]. The basic bitmap index is only effective on low-cardinality variables that have a small number of distinct values, such as the gender of a customer or the state of a customer's address. However, scientific data usually contains variables with very high cardinalities, such as floating-point values that hardly ever repeat. In Section 2 of this paper, we review key techniques to make bitmap indexes efficient on high-cardinality variables.

The software containing our bitmap indexes is called FastBit[1]. Since its release in 2007, it has been downloaded thousands of times, received an R&D 100 award, contributed to a number of PhD theses [19, 24]. It has also been integrated with a number of popular data management and analysis systems, such as HDF5 [31, 13] and VisIt [7, 22]. Many such work are performed without direct involvement from any FastBit developer [23, 26]. For example, a group in Brazil used FastBit to develop an indexing method for geographical data warehouses that was 10 - 20 times faster than existing techniques [26]. A group in Germany developed a drug screening software that was 100s of times faster than competing versions [23]. In Section 3 of this paper, we review three use cases that do involve the developers. These applications use very distinct features of FastBit to locate individual records satisfying specified conditions (Sec. 3.1), to find connected records (Sec. 3.2), and to compute conditional histograms (Sec. 3.3).

2   Core Bitmap Indexing Techniques

In this section we review the core bitmap indexing techniques used in FastBit. We start with a brief description of the basic bitmap index.

2.1   Basic bitmap index

Figure 1 shows a logical view of the basic bitmap index. Conceptually, this index contains the same information as a B-tree [8, 18]. The key difference is that a B-tree would store a list of Row IDentifiers (RIDs) for each distinct value of column **A**, whereas a bitmap index represents the same information as sequences of bits, which we call *bitmaps*. In this basic bitmap index, each bitmap corresponds to a particular value of the column. A bit with value 1 indicates that a particular row has the value represented by the bitmap.

The bitmap indexes are particularly useful for query-intensive applications, such as data warehousing and on-line analytical processing. One of the key reasons is that queries can be answered with bitwise logical operations on the bitmaps. In the example shown in Figure 1, a query "**A** < 2" can be answered by performing bitwise OR on b_1 and b_2 (i.e., $b_1 \mid b_2$). Since most computer hardware support such bitwise logical operations efficiently, the queries can be answered quickly in general. Another key reason is that answers from different bitmap indexes can be easily combined. This is because the answer from each bitmap index is a bitmap and combining the different answers simply requires additional bitwise logical operations. Since combining answers from different

---

[1]FastBit software is available from `https://codeforge.lbl.gov/projects/fastbit/`.

		bitmap index			
RID	**A**	=0	=1	=2	=3
1	0	1	0	0	0
2	1	0	1	0	0
3	2	0	0	1	0
4	2	0	0	1	0
5	3	0	0	0	1
6	3	0	0	0	1
7	1	0	1	0	0
8	3	0	0	0	1
		b_1	b_2	b_3	b_4

Figure 1: An illustration of the basic bitmap index for a column **A** that can only take on four distinct values from 0 to 3. Note RID is the short-hand for "row identifiers".

indexes swiftly is such an important consideration, a number of DBMS that do not support bitmap indexes, such as PostgreSQL and MS SQL Server, even convert intermediate solutions to bitmaps so they can be combined them more effectively.

The biggest weakness of the basic bitmap index is that its size grows linearly with the number of distinct values. Next we review different strategies to control the index sizes and improve the query response time, namely, compression, encoding and binning.

2.2 Bitmap Compression

Each bitmap in a bitmap index usually contains a large number of 0s, making it easy to compress. To answer queries efficiently, researchers have developed specialized compression methods that offer faster bitwise logical operations and consequently faster query response time [16, 37, 38]. The most widely used bitmap compression method is Byte-aligned Bitmap Code (BBC) [2]. We developed a new bitmap compression method called Word-Aligned Hybrid (WAH) code and showed it to perform bitwise logical operations more than 10 times faster than BBC [35, 36].

WAH gains its speed partly from its simplicity. For long sequences of 0s or 1s, it uses run-length encoding to represent them and for relatively short sequences of mixed 0s and 1s, it represents the bits literally. Hence, it is a hybrid of two methods. Another key feature that enables it to achieve performance is that the compressed data is word-aligned. More specifically, WAH compressed data contains two types of words; *literal* words and *fill* words.

As illustrated in Figure 2(a), a *literal* word contains one bit to indicate its type and uses the remaining bits to store the bitmap literally. A *fill* word similarly needs 1 bit to indicate its type. It uses another bit to indicate whether the bits are all 0s or all 1s, and the remaining bits are used to store the number of bits in a bitmap it represents. The number of bits represented by a WAH *fill* word is always a multiple of the number of bits stored in a literal word.

Figure 2(b) illustrates the effects of compression on overall query response time. In this test, the commercial implementation of the bitmap index (marked as "DBMS bitmap index") uses BBC compression, while "FastBit index" uses WAH compression. The query response time reported are average time values over thousands of ad hoc range queries that produce the same number of hits. Overall, the WAH compressed indexes answer queries about 14 times faster than the commercial bitmap indexes.

2.3 Bitmap Encoding

Bitmap encoding techniques manipulate the bitmaps produced by the basic bitmap index to reduce either the number of bitmaps in an index or the number of bitmaps needed to answer a query. For

3

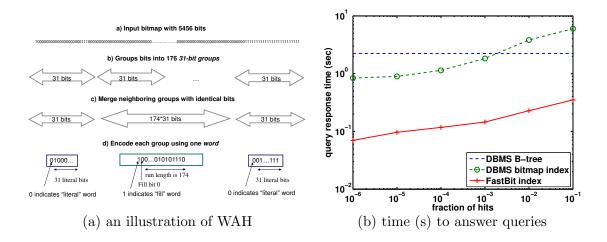| a) an illustration of WAH | (b) time (s) to answer queries |

Figure 2: The effects of compression on query response time. The faster WAH compression used in FastBit reduces the query response time by an order of magnitude (Timing measurements collected with a 400 MHz UltraSPARC II CPU [36, 29]).

example, to answer a one-sided range query of the form "$A < 3$" on the data shown in Figure 1, one needs to OR three bitmaps b_1, b_2 and b_3. It is possible to store C bitmaps that correspond to $A \le a_i$ for each of the C distinct values of A, where C is called the *column cardinality* of A. Such a bitmap index would have the same number of bitmaps as the basic bitmap index, but can answer each one-sided range query by reading just one bitmap. This is the *range encoding* [4]. Along with the *interval encoding* and the *equality encoding* (used in the basic bitmap index), there are three basic bitmap encoding methods. Next, we briefly explore strategies to compose more advanced encodings using these basic methods.

The two common strategies of composing bitmap encodings are multi-component encoding [4] and multi-level encoding [34, 25]. The central idea of multi-component encoding is to break the key value corresponding to a bitmap into multiple components in the same way an integer number is broken into multiple digits in a decimal or binary representation. In general, using more components reduces the number of bitmaps needed, which may reduce the index size. Carrying this to the extreme, we can make all base sizes as small as possible (where the minimum is 2). This particular multi-component encoding can be optimized to be the binary encoding. This encoding produces the minimum number of bitmaps and the corresponding index size is the smallest without compression.

The new class of encoding method we proposed was the multi-level index composed of a hierarchy of nested bins on the base data. Each level in such an index is a complete index on its own, and can answer queries on its own or in combination with other levels. This flexibility can be exploited to achieve better performance [41]. Figure 3 shows some timing measurements of five multi-level and multi-component indexes. These timing measurements indicate that two-level encodings, especially the range-equality encoding RE and the interval-equality encoding IE, can be as much as ten times faster than the basic bitmap index (marked E1). On average, the two-level encoded indexes are about 3 to 5 times faster than the basic bitmap index and the binary encoded index (BN).

2.4 Binning

Scientific data are often computed or collected with high precision. For example, the temperature and pressure in a combustion simulation are computed with 64-bit floating-point values having 16 decimal digits of precision. To capture the full precision of the data in a bitmap index, one typically needs millions of bitmaps in the basic bitmap index. Such indexes can be large and slow to work
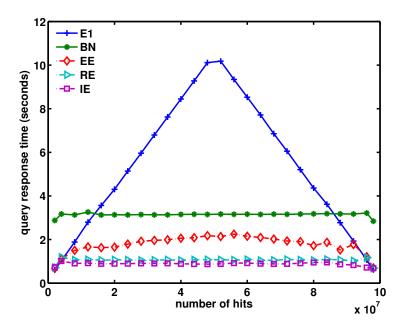
4

Figure 3: The average query response time of five different bitmap encoding methods with WAH compression (BN: binary encoding, E1: the basic one-component equality encoding, EE: two-level equality-equality encoding, RE: two-level range-equality encoding, IE: two-level interval-equality encoding; timing measurements collected with a 2.8 GHz Pentium 4 [41]).

with, even with the best compression and bitmap encoding techniques. However, we observe that an exploratory process does not usually query data at the full precision. For example, a typical query involving pressure is of the form "pressure $> 2 \times 10^7$ Pascal." In this case, the constant in the query expression has only one significant digit. To answer all such queries with one-significant-digit constants, we only need to make sure all the values are binned to one significant digit, which only requires 9 bins for each distinct exponent in the data. Even if the user requires more than one significant digit, typically, 2 or 3 significant digits are enough. The number of bins needed are still modest.

Binning can reduce the number of bitmaps in an index and improve the query response time as well. However, for some queries, we have to go back to the base data in order to answer the queries accurately. For example, if we have 99 equal-width bins for pressure between 0 and 10^9 (with 100 bin boundaries at $i \times 10^7, i = 0, \ldots, 100$), then the query condition "pressure $> 2.5 \times 10^7$" cannot be resolved with the index only. More specifically, bins 0 and 1 contain records that satisfy the query condition, and bins 3 and onwards contain records that do not satisfy the condition, but records in bin 2 (representing $2 \times 10^7 \leq$ pressure $< 3 \times 10^7$) need to be examined further. In this case, we say the records in bin 2 are *candidates* of the query. The process of determining which candidate actually satisfies the condition is called a *candidate check*. When a candidate check is needed, it often dominates the total query response time.

We have studied different approaches to minimize the impact of candidate checks. One is to reorder the expression being evaluated to minimize the overall cost of candidate checks for a query [30]. Another approach is to place the bin boundaries to minimize the cost of evaluating a fixed set of queries [17, 20]. More recently, we proposed a new approach to more directly reduce the candidate check cost by providing a clustered copy of the base data named Order-preserving Bin-based Clustering (OrBiC) [40]. This approach organizes all values of each bin contiguously on disk to reduce the I/O time needed for candidate checks. Figure 4 shows some performance numbers to illustrate the advantages of this new approach. Figure 4(a) shows the binned index with OrBiC outperforms the one without OrBiC for all query condition tested, and Figure 4(b)

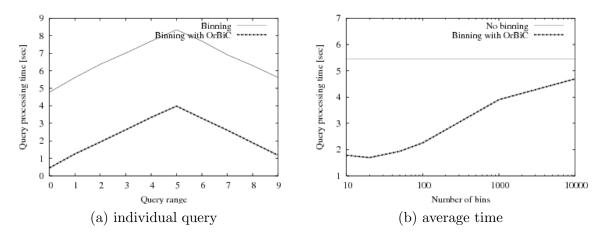5

(a) individual query   (b) average time

Figure 4: Time needed to process range queries using indexes with and without Order-preserving Bin-based Clustering (OrBiC) (Timing measurements collected on a 2.8 GHz Pentium 4 [40]).

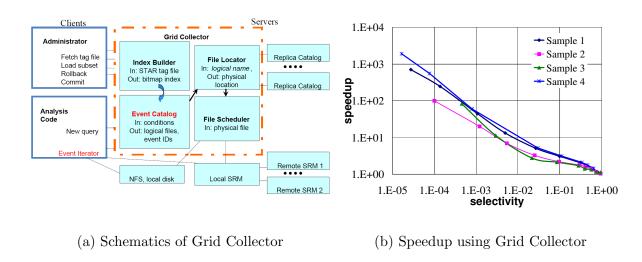(a) Schematics of Grid Collector   (b) Speedup using Grid Collector

Figure 5: Using Grid Collector can significantly speed up analysis of STAR data (Timing measurements collected on a cluster with 2.8 GHz Intel Xeon processors [32, 42]).

shows the average time used by a binned index with OrBiC to be less than that used by the precise index (with no binning).

3 Extended Technologies to Support Applications

After reviewing the core techniques behind the FastBit searching capability, we now show how this basic searching capability enables more complex searching operations.

3.1 Smart Parallel Iterator

A large data set is often organized as many files each containing a number of data records, and the files are usually stored on massive storage systems. To perform an analysis, one needs a cluster of computers to hold the necessary files. The analysis system needs to retrieve and replicate the files on the cluster. Managing the file replication and extracting the desired records can be challenging. The mapReduce systems from google and Yahoo! automate some of these operations [9]. The core concept in such a system is the parallel iterator that coordinates the accesses to files and records.
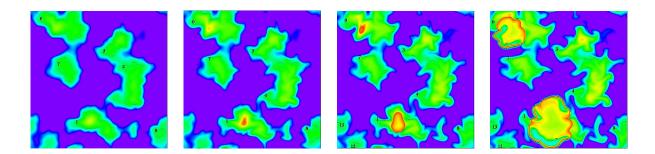
Figure 6: Tracking the evolution of ignition kernels in a combustion simulation [33].

In this example, we demonstrate how FastBit fits into an efficient parallel iterator system.

The version of parallel iterator system we built was called Grid Collector. The prototype was used in the STAR experiment and integrated into the STAR Data Analysis Framework [14, 42]. The parallel iterator extended the functionality of the Event Iterator in the STAR analysis framework, and could be used transparently with existing analysis programs. Figure 5(a) shows the schematics of Grid Collector. FastBit is responsible for building the indexes for the Event Catalog and resolving the selection criteria specified by the users to identify files and events. The Grid Collector server also scheduled the file movements and cached files with the storage resource managers.

Figure 5(b) shows the observed speedup values versus the selectivity of the analysis task, where the baseline is the time needed to run the same analysis without Grid Collector. The selectivity is the fraction of events in these files that are selected for the analysis task. The Grid Collector speeds up analysis tasks primarily by reducing the amount of disk pages accessed. When one out of 1000 events is selected, the speedup values are observed to be between 20 and 50. When one in ten events is used in an analysis job, the observed speedup is more than 2-fold.

3.2 Tracking spatial features

The previous example contains independent data records; however, many others have correlated records and the search operations depend on this correlation. For example, the records produced by a combustion simulation are related to each other by a mesh, to find and track ignition kernels, we need the connectivity information of this underlying mesh (see Figure 6) [33, 28]. FastBit indexes are uniquely able to take advantage of the underlying correlations among the data records because they preserve the ordering of data provided by the user.

We divide the task of tracking ignition kernels into three steps: 1 finding points (records) satisfying the user provided conditions for ignition kernels, 2 grouping the points into connected components, and 3 tracking the evolution of the components by computing the overlap among them. On a casual examination, FastBit indexes seem to only offer help in the first step. Indeed, many database indexes can only accelerate the first step. However, FastBit indexes and the underlying compressed bitmap data structure can speed up all three steps.

In step 2, we are able to take advantage of the compressed answers generated from step 1 to provide a compact representation of the points in the regions of interest. Instead of recording each data point separately, we store *line segments* consisting of multiple consecutive points [33]. We further provide an efficient union-find data structure to accelerate step 2 [39]. At the end of step 2, we again represent each connected component as a compressed bitmap, which allows step 3 to be completed quickly with our efficient bitwise AND operations.

In general, the first two steps can find any regions of interest. We observed that the time needed is a linear function of the number of line segments [33]. Since the number of line segments is bounded by the number of points on the surface of the regions, our algorithm for finding regions of interest is bounded by a linear function of the number of points on the surface, instead of the

7

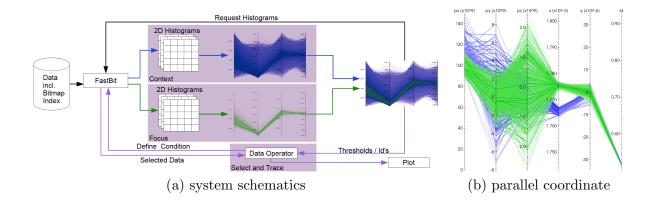(a) system schematics   (b) parallel coordinate

Figure 7: The schematics of the parallel coordinate system used for selecting and displaying particles from a Laser Wake-Field Accelerator simulation (a) and a parallel coordinate display for a time step (b) [22].

number of points in the regions. For well-defined regions, where the number of point inside is much larger than the number of points on the surface, our approach has a significant advantage.

3.3   Dynamic conditional histograms

Aside from its searching capability, FastBit also has a set of efficient functions for computing conditional histograms [27, 22, 21]. In this example of visual exploration of Laser Wake-Field Accelerator (LWFA) simulation data, we illustrate how this capability is used to produce parallel coordinate plots and facilitate the tracking and understanding of particle bundles in LWFA[2]. The new visual analysis system reduced the turn-around time from hours to seconds.

Figure 7(a) illustrates the use of FastBit to produce a parallel coordinate plot in this application. FastBit is used to select records according to user specified conditions and to compute the histograms on the selected records. For example, the context in Figure 7(b) (shown in blue) was selected with "px $> 8.872 \times 10^{10}$." The focus (shown in green) was selected with an additional condition of "x $> 1.77 \times 10^{-3}$." The parallel coordinate plots contain only particles with a high forward momentum px (shown as the first axis on the left) $> 8.872 \times 10^{10}$. These particles form two well-separated groups on the x-axis (shown as the 4th axis from the left), an important feature for understanding the particle bundles in LWFA.

These parallel coordinate plots are generated using 2-D histograms involving neighboring axes. FastBit produced these histograms with the data records satisfying the user specified conditions and removed the need for user (or the visualization system) to read the whole data set before perform the filtering. This reduced the volume of data accessed and improved the interactivity of the visual exploration [22].

In the same application, FastBit was also used to track particles through time and select data for volume rendering and other computations. For example, after a localized group of accelerated particles was found in one time step, scientists might track its history to understand how the group was formed. To accomplish this task, we first extracted the identifiers of the particles, then found particles with these identifiers in other time steps. This search operation involved many thousands of particle identifiers over millions of particles in the base data. The ability to handle this operation efficiently is another feature of FastBit.

---

[2]Note that the user data is stored in HDF5 files [31, 13] and visualization is performed with VisIt [7].

8

4 Summary and Future Plans

FastBit employs a number of advanced bitmap indexing techniques to significantly reduce the query response time on large scientific data. Its advantages have not only been observed in timing measurements, but also proven through theoretical computational complexity analyses [37, 38]. Furthermore, we have extended FastBit to include functions for handling simple mesh information and computing conditional histograms. All these are useful features in analyzing massive data sets. In addition, FastBit software is available open source so that anyone can derive from it or extend it for their own work.

A number of new features are currently planned for FastBit, such as handling of queries that join multiple tables. To support more applications, we are working on integrating information about irregular and semi-regular meshes for queries involving spatial data. We are also studying ways to perform text searches using compressed bitmap indexes.

FastBit shares a number of shortcomings of all other bitmap indexes; the most important of which is that they are slow to update after modifications of a few existing records. This prevents it from being used effectively in applications where data records change frequently. However, if the changes are limited to relatively new records, FastBit could work effectively by avoiding indexing those new records.

Acknowledgments

References

[1] T. Anticic, V. Barroso, F. Carena, W. Carena, S. Chapeland, O. Cobanoglu, E. Denes, R. Divia, U. Fuchs, T. Kiss, I. Makhlyueva, F. Ozok, F. Roukoutakis, K. Schossmaier, C. Soos, P. Vande Vyvre, and S. Vergara. Commissioning of the ALICE data acquisition system. In *CHEP'07*. IOP Publishing, 2008.

[2] G. Antoshenkov and M. Ziauddin. Query processing and optimization in ORACLE RDB. *VLDB Journal*, 5:229–237, 1996.

[3] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[4] C.-Y. Chan and Y. E. Ioannidis. Bitmap index design and evaluation. In SIGMOD'99, pages 355–366, New York, NY, USA, 1998. ACM press.

[5] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26(1):65–74, March 1997.

[6] J. H. Chen, A. Choudhary, D. De Supinsky, M DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. Mellor-Crummey, N. Podhorszki, R. Sankaran, S. Shende, and C. S. Yoo.

Terascale direct numerical simulations of turbulent combustion using S3D. *Computational Science & Discovery*, 2:015001–015031, 2009.

[7] Hank Childs, Eric S. Brugger, Kathleen S. Bonnell, Jeremy S Meredith, Mark Miller, Brad J Whitlock, and Nelson Max. A contract-based system for large data visualization. In *IEEE Visualization 2005*, pages 190–198, October 2005. Software available at http://wci.llnl.gov/codes/visit/.

[8] Douglas Comer. The ubiquitous B-tree. *Computing Surveys*, 11(2):121–137, 1979.

[9] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI'04*, 2004.

[10] Barry A. Devlin and Paul T. Murphy. An architecture for a business and information system. *IBM Systems Journal*, 27(1):60–80, 1988.

[11] S. Gameiro, G. Crone, R. Ferrari, D. Francis, B. Gorini, M. Gruwe, M. Joos, G. Lehmann, L. Mapelli, A. Misiejuk, E. Pasqualucci, J. Petersen, R. Spiwoks, L. Tremblet, G. Unel, W. Vandelli, and Y. Yasu. The ROD crate DAQ software framework of ATLAS data acquisition system. *IEEE Transactions on Nuclear Science*, 53(3):907–911, 2006.

[12] C. Geddes, C. Toth, J. van Tilborg, E. Esarey, C. Schroeder, D. Bruhwiler, C. Nieter, J. Cary, and W. Leemans. High-quality electron beams from a laser wakefield accelerator using plasma-channel guiding. *Nature*, 438:538–541, 2004.

[13] Luke Gosink, John Shalf, Kurt Stockinger, Kesheng Wu, and Wes Bethel. HDF5-FastQuery: Accelerating complex queries on HDF datasets using fast bitmap indices. In *SSDBM 2006*. IEEE Computer Society Press., 2006.

[14] J.W. Harris and STAR collaboration. The star experiment at the relativistic heavy ion collider. *Nuclear Physics A*, 566:277 – 285, 1994.

[15] William H. Inmon. The data warehouse and data mining. *Communications of the ACM*, 39(11):49–50, November 1996.

[16] Theodore Johnson. Performance measurements of compressed bitmap indices. In M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, editors, *VLDB'99*, pages 278–289, San Francisco, 1999. Morgan Kaufmann.

[17] Nick Koudas. Space efficient bitmap indexing. In *CIKM 2000*, pages 194–201. ACM, 2000.

[18] P. O'Neil. Model 204 architecture and performance. In *2nd International Workshop in High Performance Transaction Systems, Asilomar, CA*, pages 40–59. Springer-Verlag, September 1987.

[19] Frederick Ralph Reiss. *Data Triage*. PhD thesis, EECS Department, University of California, Berkeley, June 2007.

[20] Doron Rotem, Kurt Stockinger, and Kesheng Wu. Optimizing candidate check costs for bitmap indices. In *CIKM*, 2005.

[21] Oliver Rübel, Prabhat, Kesheng Wu, Hank Childs, Jeremy Meredith, Cameron G. R. Geddes, Estelle Cormier-Michel, Sean Ahern, Gunther H. Weber, Peter Messmer, Hans Hagen, Bernd Hamann, and E. Wes Bethel. Application of high-performance visual analysis methods to laser wakefield particle acceleration data. In *IEEE Visualization 2008*, Columbus, Ohio, USA, October 2008. LBNL-952E.

[22] Oliver Rübel, Prabhat, Kesheng Wu, Hank Childs, Jeremy Meredith, Cameron G. R. Geddes, Estelle Cormier-Michel, Sean Ahern, Gunther H. weber, Peter Messmer, Hans Hagen, Bernd Hamann, and E. Wes Bethel. High performance multivariate visual data exploration for extemely large data. In *SuperComputing 2008 (SC08)*, page 51, Austin, Texas, USA, November 2008. LBNL-716E.

[23] Jochen Schlosser and Matthias Rarey. Beyond the virtual screening paradigm: Structure-based searching for new lead compounds. *Journal of Chemical Information and Modeling*, 49(4):800–809, 2009.

[24] Rishi Sinha. *Indexing Scientific Data*. PhD thesis, UIUC, 2007.

[25] Rishi Rakesh Sinha and Marianne Winslett. Multi-resolution bitmap indexes for scientific data. *ACM Trans. Database Syst.*, 32(3):16, 2007.

[26] Thiago Luís Lopes Siqueira, Ricardo Rodrigues Ciferri, Valéria Cesário Times, and Cristina Dutra de Aguiar Ciferri. A spatial bitmap-based index for geographical data warehouses. In *SAC '09*, pages 1336–1342, New York, NY, USA, 2009. ACM.

[27] Kurt Stockinger, E. Wes Bethel, Scott Campbell, Eli Dart, and Kesheng Wu. Detecting distributed scans using high-performance query-driven visualization. In *SC '06*, page 82, New York, NY, USA, 2006. ACM.

[28] Kurt Stockinger, John Shalf, Wes Bethel, and Kesheng Wu. Query-driven visualization of large data sets. In *IEEE Visualization*, 2005. LBNL report LBNL-57511.

[29] Kurt Stockinger and Kesheng Wu. *Bitmap Indices for Data Warehouses*, pages 179–202. Idea Group, Inc., 2006. LBNL-59952.

[30] Kurt Stockinger, Kesheng Wu, and Arie Shoshani. Evaluation strategies for bitmap indices with binning. In *DEXA*. Springer-Verlag, September 2004.

[31] The HDF Group. HDF5 user guide, 2007.

[32] Kesheng Wu, Junmin Gu, Jerome Lauret, Arthur M. Poskanzer, Arie Shoshani, Alexander Sim, and Wei-Ming Zhang. Grid collector: Facilitating efficient selective access from data grids. LBNL report LBNL-57677.

[33] Kesheng Wu, Wendy Koegler, Jacqueline Chen, and Arie Shoshani. Using bitmap index for interactive exploration of large datasets. In *SSDBM'03*, pages 65–74, Cambridge, MA, USA, 2003.

[34] Kesheng Wu, Ekow Otoo, and Arie Shoshani. Compressed bitmap indices for efficient query processing. Technical Report LBNL-47807, Lawrence Berkeley National Laboratory, Berkeley, CA, 2001.

[35] Kesheng Wu, Ekow Otoo, and Arie Shoshani. A performance comparison of bitmap indexes. In *CIKM'01*, pages 559–561, 2001.

[36] Kesheng Wu, Ekow Otoo, and Arie Shoshani. Compressing bitmap indexes for faster search operations. In *SSDBM'02*, pages 99–108, Edinburgh, Scotland, 2002. LBNL-49627.

[37] Kesheng Wu, Ekow Otoo, and Arie Shoshani. On the performance of bitmap indices for high cardinality attributes. In *VLDB*, pages 24–35, 2004.

[38] Kesheng Wu, Ekow Otoo, and Arie Shoshani. Optimizing bitmap indices with efficient compression. *ACM Transactions on Database Systems*, 31:1–38, 2006.

[39] Kesheng Wu, Ekow Otoo, and Kenji Suzuk. Optimizing two-pass connected-component labeling algorithms. *Pattern Analysis & Applications*, 12(2):117–135, 2009.

[40] Kesheng Wu, Kurt Stockinger, and Arie Shosani. Breaking the curse of cardinality on bitmap indexes. In *SSDBM'08*, pages 348–365, 2008.

[41] Kesheng Wu, Kurt Stockinger, and Arie Shoshani. Performance of multi-level and multi-component compressed bitmap indexes. Technical Report LBNL-60891, Lawrence Berkeley National Laboratory, Berkeley, CA, 2007.

[42] Kesheng Wu, Wei-Ming Zhang, Victor Perevoztchikov, Jerome Lauret, and Arie Shoshani. The grid collector: Using an event catalog to speed up user analysis in distributed environment. In *CHEP2004*, 2004.

[43] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2):6, 2006.